RUBY BUILT-IN FUNCTIONS

Since the *Kernel* module is included by *Object* class, its methods are available everywhere in the Ruby program. They can be called without a receiver (functional form). Therefore, they are often called functions.

A complete list of Built-in Functions is given here for your reference:

SN	Methods with Description
1	**abort** Terminates program. If an exception is raised (i.e., $! isn't nil), its error message is displayed.
2	**Array(obj)** Returns obj after converting it to an array using to_ary or to_a.
3	**at_exit {...}** Registers a block for execution when the program exits. Similar to END statement, but END statement registers the block only once.
4	**autoload(classname, file)** Registers a class classname to be loaded from file the first time it's used. classname may be a string or a symbol.
5	**binding** Returns the current variable and method bindings. The *Binding* object that is returned may be passed to the *eval* method as its second argument.
6	**block_given?** Returns true if the method was called with a *block*.
7	**callcc {| c|...}** Passes a *Continuation* object c to the block and executes the block. *callcc* can be used for global exit or loop construct.
8	**caller([n])** Returns the current execution stack in an array of the strings in the form *file:line*. If n is specified, returns stack entries from nth level on down.
9	**catch(tag) {...}** Catches a nonlocal exit by a throw called during the execution of its block.
10	**chomp([rs=$/])** Returns the value of variable $_ with the ending newline removed, assigning the result back to $_. The value of the newline string can be specified with rs.
11	**chomp!([rs=$/])** Removes newline from $_, modifying the string in place.
12	**chop** Returns the value of $_ with its last character (one byte) removed, assigning the result back to $_.

13	**chop!** Removes the last character from $_, modifying the string in place.
14	**eval(str[, scope[, file, line]])** Executes *str* as Ruby code. The binding in which to perform the evaluation may be specified with *scope*. The filename and line number of the code to be compiled may be specified using *file* and line.
15	**exec(cmd[, arg...])** Replaces the current process by running the command *cmd*. If multiple arguments are specified, the command is executed with no shell expansion.
16	**exit([result=0])** Exits program, with *result* as the status code returned.
17	**exit!([result=0])** Kills the program bypassing exit handling such as *ensure*, etc.
18	**fail(...)** See raise(...)
19	**Float(obj)** Returns obj after converting it to a float. Numeric objects are converted directly; nil is converted to 0.0; strings are converted considering 0x, 0b radix prefix. The rest are converted using obj.to_f.
20	**fork** **fork {...}** Creates a child process. *nil* is returned in the child process and the child process' ID (integer) is returned in the parent process. If a block is specified, it's run in the child process.
21	**format(fmt[, arg...])** See sprintf.
22	**gets([rs=$/])** Reads the filename specified in the command line or one line from standard input. The record separator string can be specified explicitly with rs.
23	**global_variables** Returns an array of global variable names.
24	**gsub(x, y)** **gsub(x) {...}** Replaces all strings matching x in $_ with y. If a block is specified, matched strings are replaced with the result of the block. The modified result is assigned to $_.
25	**gsub!(x, y)** **gsub!(x) {...}** Performs the same substitution as gsub, except the string is changed in place.
26	**Integer(obj)** Returns obj after converting it to an integer. Numeric objects are converted directly; nil is converted to 0; strings are converted considering 0x, 0b radix prefix. The rest are converted using obj.to_i.
27	**lambda {| x|...}** **proc {| x|...}**

	lambda **proc** Converts a block into a *Proc* object. If no block is specified, the block associated with the calling method is converted.
28	**load(file[, private=false])** Loads a Ruby program from *file*. Unlike *require*, it doesn't load extension libraries. If *private* is *true*, the program is loaded into an anonymous module, thus protecting the namespace of the calling program.
29	**local_variables** Returns an array of local variable names.
30	**loop {...}** Repeats a block of code.
31	**open(path[, mode="r"])** **open(path[, mode="r"]) {| f|...}** Opens a *file*. If a block is specified, the block is executed with the opened stream passed as an argument. The file is closed automatically when the block exits. If *path* begins with a pipe |, the following string is run as a command, and the stream associated with that process is returned.
32	**p(obj)** Displays obj using its inspect method (often used for debugging).
33	**print([arg...])** Prints arg to *$defout*. If no arguments are specified, the value of $_ is printed.
34	**printf(fmt[, arg...])** Formats arg according to *fmt* using *sprintf* and prints the result to *$defout*. For formatting specifications, see sprintf for detail.
35	**proc {| x|...}** **proc** See lamda.
36	**putc(c)** Prints one character to the default output (*$defout*).
37	**puts([str])** Prints string to the default output (*$defout*). If the string doesn't end with a newline, a newline is appended to the string.
38	**raise(...)** **fail(...)** Raises an exception. Assumes *RuntimeError* if no exception class is specified. Calling *raise* without arguments in a *rescue* clause re-raises the exception. Doing so outside a rescue clause raises a message-less *RuntimeError*. **fail** is an obsolete name for raise.
39	**rand([max=0])** Generates a pseudo-random number greater than or equal to 0 and less than max. If max is either not specified or is set to 0, a random number is returned as a floating-point number greater than or equal to 0 and less than 1. *srand* may be used to initialize pseudo-random stream.
40	**readline([rs=$/])**

	Equivalent to gets except it raises an EOFError exception on reading EOF.
41	**readlines([rs=$/])** Returns an array of strings holding either the filenames specified as command-line arguments or the contents of standard input.
42	**require(lib)** Loads the library (including extension libraries) *lib* when it's first called. require will not load the same library more than once. If no extension is specified in *lib*, require tries to add .rb,.so, etc., to it.
43	**scan(re)** **scan(re) {|x|...}** Equivalent to $_.scan.
44	**select(reads[, writes=nil[, excepts=nil[, timeout=nil]]])** Checks for changes in the status of three types of IO objects input, output, and exceptions which are passed as arrays of IO objects. *nil* is passed for arguments that don't need checking. A three-element array containing arrays of the IO objects for which there were changes in status is returned. *nil* is returned on timeout.
45	**set_trace_func(proc)** Sets a handler for tracing. *proc* may be a string or *proc* object. *set_trace_func* is used by the debugger and profiler.
46	**sleep([sec])** Suspends program execution for sec seconds. If sec isn't specified, the program is suspended forever.
47	**split([sep[, max]])** Equivalent to $_.split.
48	**sprintf(fmt[, arg...])** **format(fmt[, arg...])** Returns a string in which arg is formatted according to fmt. Formatting specifications are essentially the same as those for sprintf in the C programming language. Conversion specifiers (% followed by conversion field specifier) in *fmt* are replaced by formatted string of corresponding argument. A list of conversion filed is given below in next section.
49	**srand([seed])** Initializes an array of random numbers. If *seed* isn't specified, initialization is performed using the time and other system information for the seed.
50	**String(obj)** Returns obj after converting it to a string using obj.to_s.
51	**syscall(sys[, arg...])** Calls an operating system call function specified by number *sys*. The numbers and meaning of *sys* is system-dependant.
52	**system(cmd[, arg...])** Executes *cmd* as a call to the command line. If multiple arguments are specified, the command is run directly with no shell expansion. Returns *true* if the return status is 0 (success).
53	**sub(x, y)** **sub(x) {...}** Replaces the first string matching x in $_ with y. If a block is specified, matched strings are replaced with the

	result of the block. The modified result is assigned to $_.
54	**sub!(x, y)** **sub!(x) {...}** Performs the same replacement as sub, except the string is changed in place.
55	**test(test, f1[, f2])** Performs various file tests specified by the character *test*. In order to improve readability, you should use File class methods (for example File::readable?) rather than this function. A list of arguments is given below in next section.
56	**throw(tag[, value=nil])** Jumps to the catch function waiting with the symbol or string *tag*. value is the return value to be used by *catch*.
57	**trace_var(var, cmd)** **trace_var(var) {...}** Sets tracing for a global variable. The variable name is specified as a symbol. *cmd* may be a string or Proc object.
58	**trap(sig, cmd)** **trap(sig) {...}** Sets a signal handler. sig may be a string (like SIGUSR1) or an integer. SIG may be omitted from signal name. Signal handler for EXIT signal or signal number 0 is invoked just before process termination.
59	**untrace_var(var[, cmd])** Removes tracing for a global variable. If *cmd* is specified, only that command is removed.

Functions for Numbers:

Here is list of Built-in Functions related to number. They should be used as follows:

```
#!/usr/bin/ruby

num = 12.40
a.floor    ==> 12
num + 10   ==> 22.40
num.integer? ==> true  as num is an integer.
```

Assuming, **n** is a number:

SN	Methods with Description
1	**n + num** **n - num** **n * num** **n / num** Performs arithmetic operations: addition, subtraction, multiplication, and division.
1	**n % num** Returns the modulus of n.
2	**n ** num**

	Exponentiation.
3	**n.abs** Returns the absolute value of n.
4	**n.ceil** Returns the smallest integer greater than or equal to n.
5	**n.coerce(num)** Returns an array containing num and n both possibly converted to a type that allows them to be operated on mutually. Used in automatic type conversion in numeric operators.
6	**n.divmod(num)** Returns an array containing the quotient and modulus from dividing n by num.
7	**n.floor** Returns the largest integer less than or equal to n.
8	**n.integer?** Returns true if n is an integer.
9	**n.modulo(num)** Returns the modulus obtained by dividing n by num and rounding the quotient with *floor*
10	**n.nonzero?** Returns n if it isn't zero, otherwise nil.
11	**n.remainder(num)** Returns the remainder obtained by dividing **n** by **num** and removing decimals from the quotient. The **result** and **n** always have same sign.
12	**n.round** Returns n rounded to the nearest integer.
13	**n.truncate** Returns n as an integer with decimals removed.
14	**n.zero?** Returns zero if n is 0.
15	**n & num** **n | num** **n ^ num** Bitwise operations: AND, OR, XOR, and inversion.
16	**n << num** **n >> num** Bitwise left shift and right shift.
17	**n[num]** Returns the value of the **num**th bit from the least significant bit, which is n[0].
18	**n.chr** Returns a string containing the character for the character code **n**

19	**n.next** **n.succ** Returns the next integer following n. Equivalent to n + 1.
20	**n.size** Returns the number of bytes in the machine representation of **n**
21	**n.step(upto, step) {|n| ...}** Iterates the block from **n** to **upto**, incrementing by **step** each time.
22	**n.times {|n| ...}** Iterates the block **n** times.
23	**n.to_f** Converts **n** into a floating point number. Float conversion may lose precision information.
24	**n.to_int** Returns **n** after converting into interger number.

Functions for Float

Here is a list of Ruby Built-in function specially for float numbers. Assuming we have a float number **f**:

SN	Methods with Description
1	**Float::induced_from(num)** Returns the result of converting *num* to a floating-point number.
2	**f.finite?** Returns true if f isn't infinite and f.nan is false.
3	**f.infinite?** Returns 1 if f is positive infinity, -1 if negative infinity, or nil if anything else.
4	**f.nan?** Returns true if f isn't a valid IEEE floating point number.

Functions for Math

Here is a list of Ruby Built-in math functions:

SN	Methods with Description
1	**atan2(x, y)** Calculates the arc tangent.
2	**cos(x)** Calculates the cosine of x.
3	**exp(x)**

	Calculates an exponential function (e raised to the power of x).
4	**frexp(x)** Returns a two-element array containing the nominalized fraction and exponent of x.
5	**ldexp(x, exp)** Returns the value of x times 2 to the power of exp.
6	**log(x)** Calculates the natural logarithm of x.
7	**log10(x)** Calculates the base 10 logarithm of x.
8	**sin(x)** Calculates the sine of x.
9	**sqrt(x)** Returns the square root of x. x must be positive.
10	**tan(x)** Calculates the tangent of x.

Conversion Field Specifier:

The function *sprintf(fmt[, arg...]) and format(fmt[, arg...])* returns a string in which arg is formatted according to fmt. Formatting specifications are essentially the same as those for sprintf in the C programming language. Conversion specifiers (% followed by conversion field specifier) in *fmt* are replaced by formatted string of corresponding argument.

The following conversion specifiers, are supported by Ruby's format:

Specifier	Description
b	Binary integer
c	Single character
d,i	Decimal integer
e	Exponential notation (e.g., 2.44e6)
E	Exponential notation (e.g., 2.44E6)
f	Floating-point number (e.g., 2.44)
g	use %e if exponent is less than -4, %f otherwise
G	use %E if exponent is less than -4, %f otherwise
o	Octal integer
s	String, or any object converted using to_s
u	Unsigned decimal integer

x	Hexadecimal integer (e.g., 39ff)
X	Hexadecimal integer (e.g., 39FF)

Following is the usage example:

```
#!/usr/bin/ruby

sprintf("%s\n", "abc")   # => "abc\n" (simplest form)
sprintf("d=%d", 42)      # => "d=42" (decimal output)
sprintf("%04x", 255)     # => "00ff" (width 4, zero padded)
sprintf("%8s", "hello")  # => " hell" (space padded)
sprintf("%.2s", "hello") # => "he" (trimmed by precision)
```

Test Function Arguments:

The function *test(test, f1[, f2])* performs one of the following file tests specified by the character *test*. In order to improve readability, you should use File class methods (for example File::readable?) rather than this function. Here are the file tests with one argument:

Argument	Description
?r	Is f1 readable by the effective uid of caller?
?w	Is f1 writable by the effective uid of caller?
?x	Is f1 executable by the effective uid of caller?
?o	Is f1 owned by the effective uid of caller?
?R	Is f1 readable by the real uid of caller?
?W	Is f1 writable by the real uid of caller?
?X	Is f1 executable by the real uid of caller?
?O	Is f1 owned by the real uid of caller?
?e	Does f1 exist?
?z	Does f1 have zero length?
?s	File size of f1(nil if 0)
?f	Is f1 a regular file?
?d	Is f1 a directory?
?l	Is f1 a symbolic link?
?p	Is f1 a named pipe (FIFO)?
?S	Is f1 a socket?
?b	Is f1 a block device?
?c	Is f1 a character device?

?u	Does f1 have the setuid bit set?
?g	Does f1 have the setgid bit set?
?k	Does f1 have the sticky bit set?
?M	Last modification time for f1.
?A	Last access time for f1.
?C	Last inode change time for f1.

File tests with two arguments are as follows:

Argument	Description
?=	Are modification times of f1 and f2 equal?
?>	Is the modification time of f1 more recent than f2 ?
?<	Is the modification time of f1 older than f2 ?
?-	Is f1 a hard link to f2 ?

Following is the usage example. Assuming test.rb exist with read, write and execute permissions:

```
#!/usr/bin/ruby

test(?r, "test.rb" )   # => true
test(?w, "test.rb" )   # => true
test(?x, "test.rb" )   # => true
```